15-319 / 15-619
Cloud Computing

Overview 8

March 15th, 2022

Reflection of Week Before Spring Break

- **Conceptual content on OLI**
  - Module 13: Storage and Network Virtualization
- **Project 3: Cloud Storage**
- **Team Project Checkpoint**
- **OPE - Spark Programming**

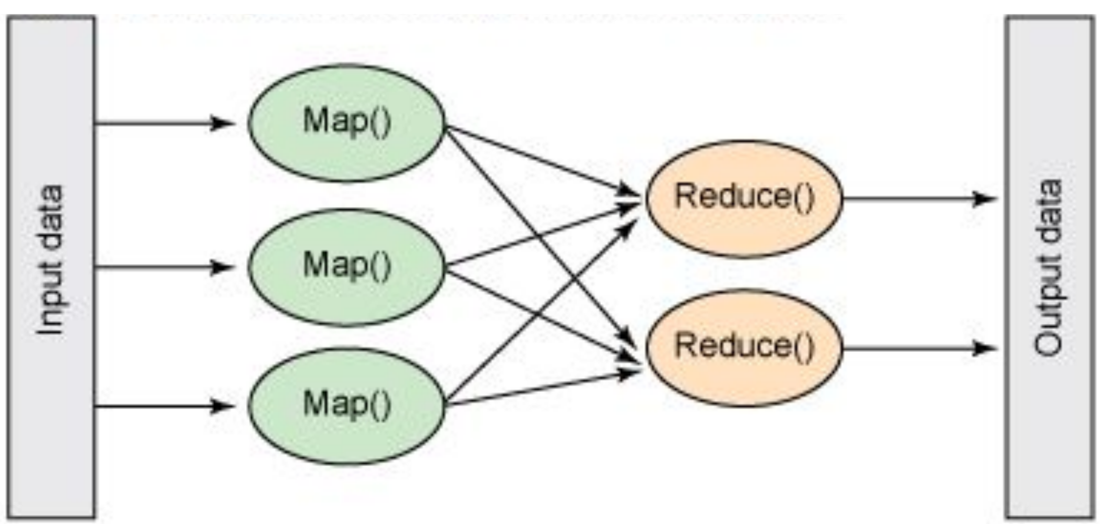
This Week

- **OLI, Unit 4: Cloud Storage**
  - Module 14: Cloud Storage
  - Module 15: Case Studies: Distributed File System
  - Module 16: Case Studies: NoSQL Databases
  - Module 17: Case Studies: Cloud Object Storage
- **Quiz 7 (OLI Module 14)**
  - Due **Friday**, March 18th, 2022, 11:59PM ET
- **Project 4 - Iterative processing with spark**
  - Due **Sunday**, March 27th, 2022, 11:59PM ET

Introduction to MapReduce

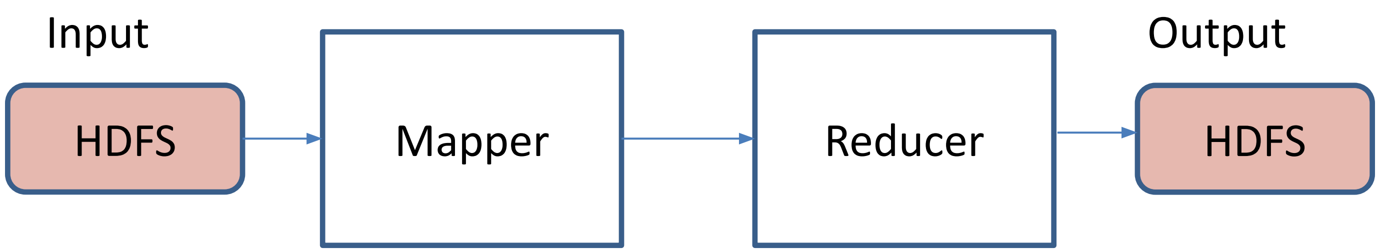
- The MapReduce programming model simplifies parallel processing by abstracting away the complexities involved in working with distributed systems

- Map: Process the input data in chunks **in parallel**
- Shuffle and sort
- Reduce: Aggregate or summarize intermediate data **in parallel** and output the result



Typical MapReduce Batch Job

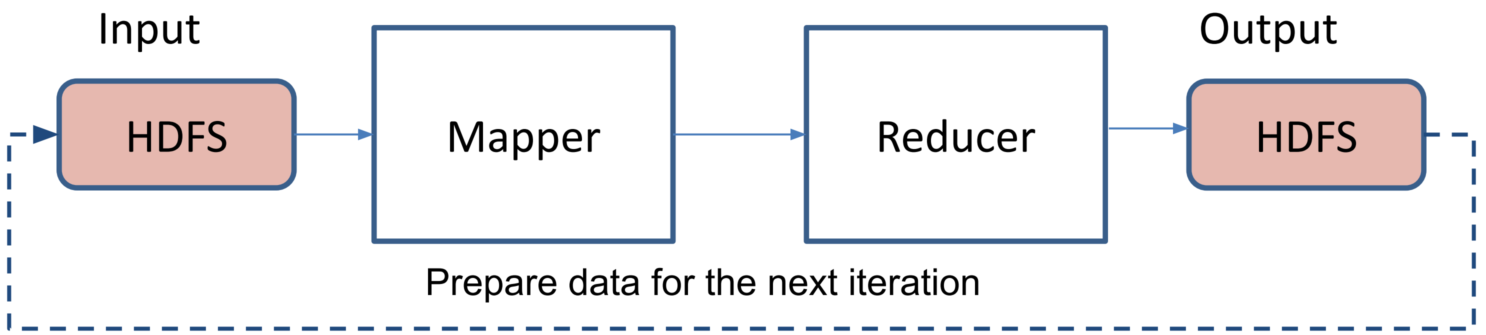
- Simplistic view of a MapReduce job



- You write code to implement the following classes
  - Mapper
  - Reducer
- Inputs are read from disk and outputs are written to disk
  - Intermediate data is spilled to local disk

Iterative MapReduce Jobs

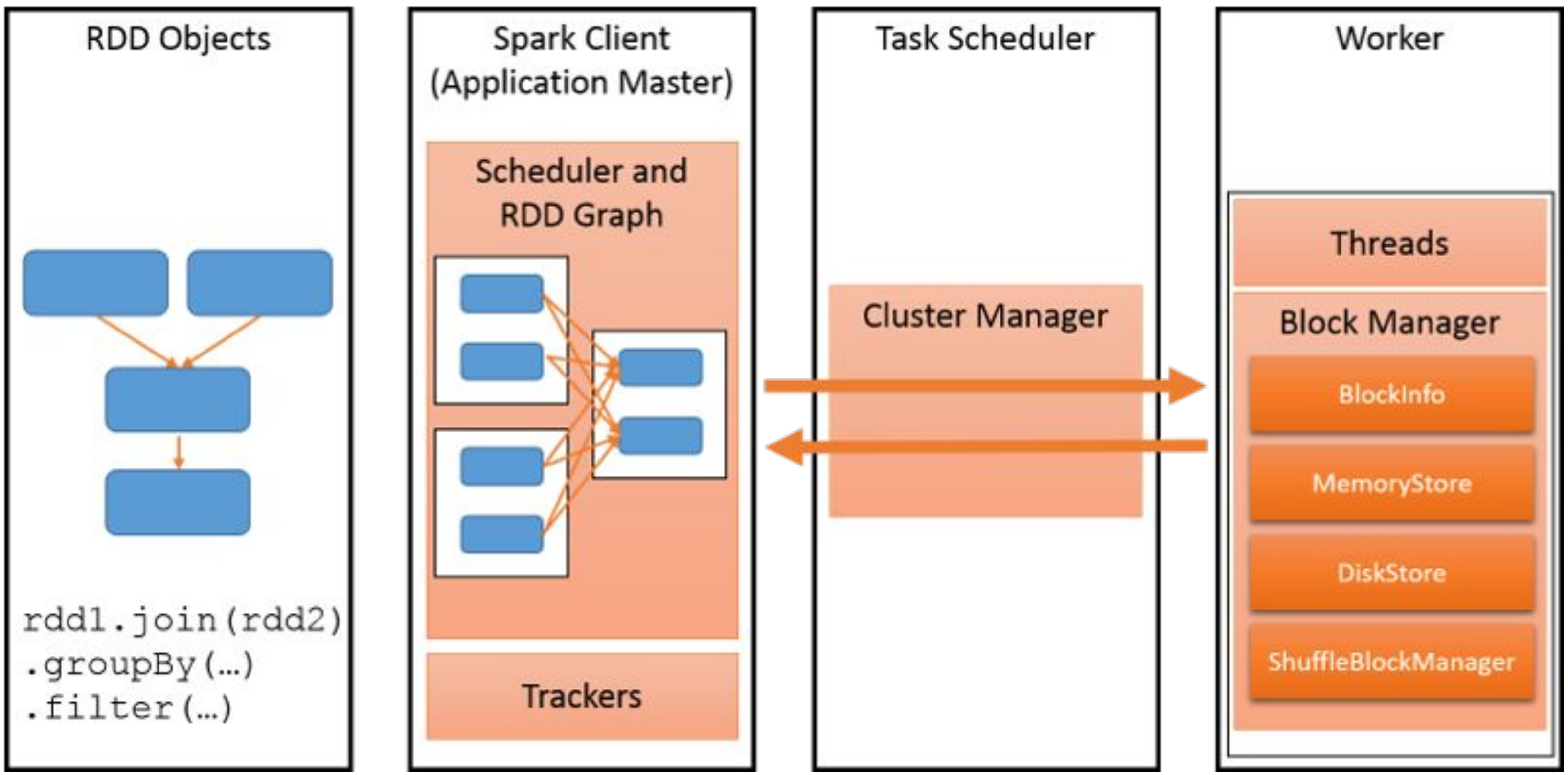
- Some applications require iterative processing
- E.g., Machine Learning



- MapReduce: Data is always **written** to disk
  – This leads to added overhead for each iteration
  – Can we keep data in memory? Across Iterations?
  – How do you manage this?

Apache Spark

- General-purpose cluster computing framework
- APIs in Python, Java, Scala and R
- Runs on Windows and UNIX-like systems



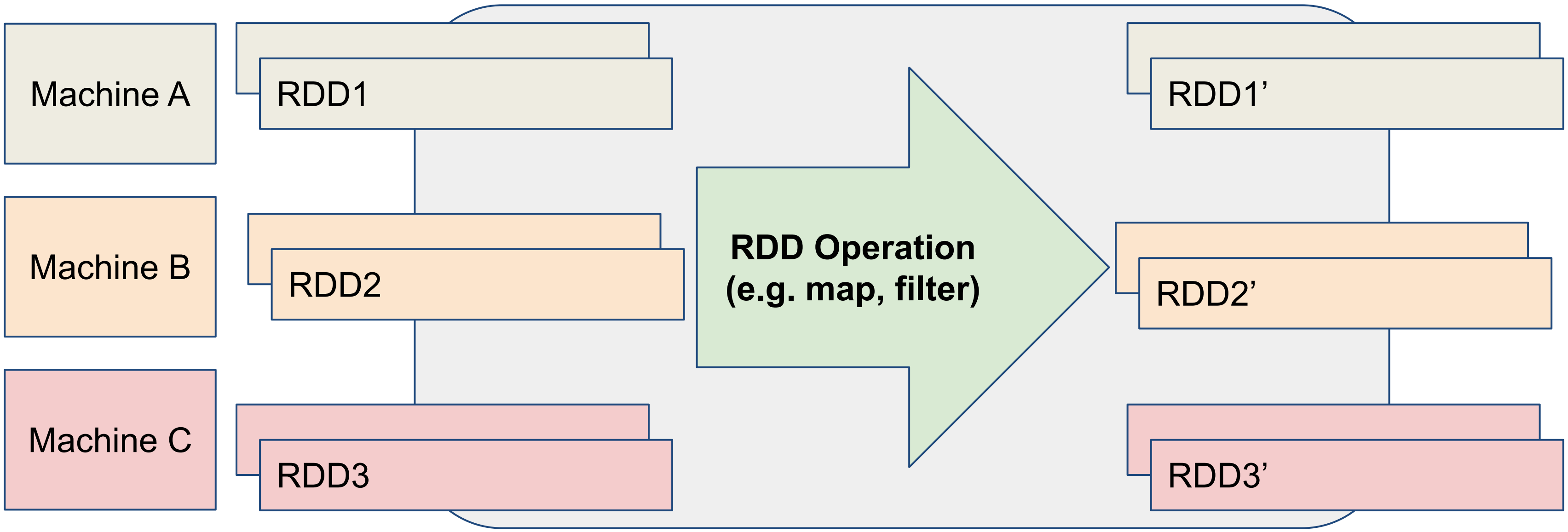
Apache Spark APIs

- There exists 3 sets of APIs for handling data in Spark

Resilient Distributed Dataset (RDD)	**DataFrame**	**Datasets**
• Distributed collection of JVM objects • Functional operators (map, filter, etc.)	• Distributed collection of Row objects • No compile time type safety • Fast, efficient internal representations	• Compile time type-safe • Fast

Key to Apache Spark - RDDs

- Resilient Distributed Datasets (RDDs)
- Can be in-memory or on disk
- Read-only objects
- Partitioned across the cluster based on a range or the hash of a key in each record



Operations on RDDs

- Loading data

```
>>> input_RDD = sc.textFile("text.file")
```

- Transformation
  - Applies an operation to derive a new RDD
  - Lazily evaluated -- may not be executed immediately

```
>>> transform_RDD =  input_RDD.filter(lambda x: "abcd" in x)
```

- Action
  - Forces the computation on an RDD
  - Returns a single object

```
>>> print "Number of “abcd”:" + transform_RDD.count()
```

- Saving data

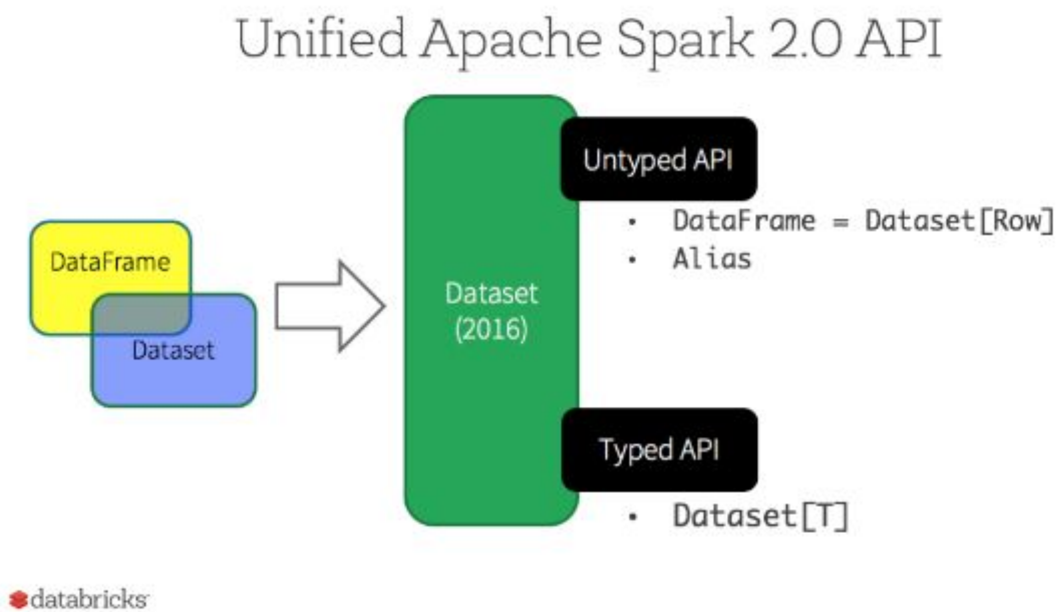
```
>>> output.saveAsTextFile(“hdfs:///output”)
```

RDDs and Fault Tolerance

- Actions create new RDDs
- Uses the notion of lineage to support fault tolerance
  - Lineage is a log of transformations
  - Stores lineage on the driver node
  - Upon node failure, Spark loads data from disk to recompute the entire sequence of operations based on lineage

DataFrames and Datasets

- A DataFrame is a collection of rows
    - Tabular
    - Organized into named columns, like a table in a relational DB
- A dataset is a collection of objects
    - Domain specific
    - Object oriented



Operations on DataFrames

- Suppose we have a file people.json

```
{"name":"Michael"} {"name":"Andy", "age":30} {"name":"Justin", "age":19}
```

- Create a DataFrame with its contents

```
val df = spark.read.json("people.json")
```

- Run SQL-like queries against the data

```
val sqlDF = df.where($"age" > 20).show()
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

- Save data to file

```
df.where($"age" > 20).select("name").write.parquet("output")
```

Note: Parquet is a column-based storage format for Hadoop.

Spark Ecosystem

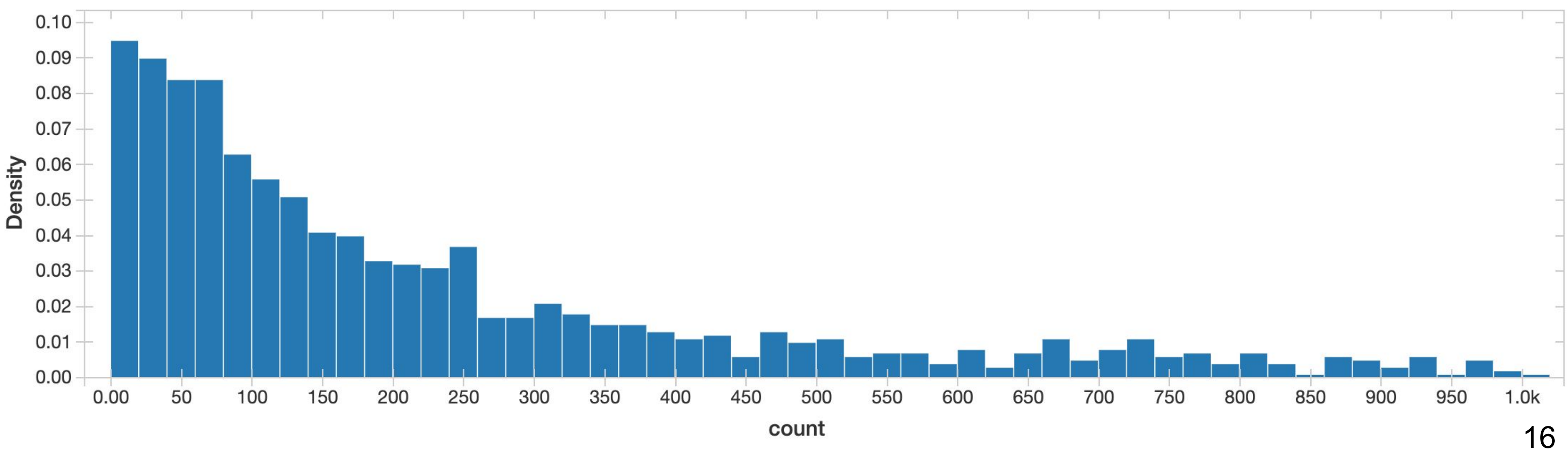
- Spark SQL
  - Process structured data
  - Run SQL-like queries against RDDs
- Spark Streaming
  - Ingest data from sources like Kafka
  - Process data with high level functions like map and reduce
  - Output data to live dashboards or databases
- MLlib
  - Machine learning algorithms such as regression
  - Utilities such as linear algebra and statistics
- GraphX
  - Graph-parallel framework
  - Support for graph algorithms and analysis

Project 4
Iterative Processing with Spark

- **Task 1:** Exploratory Analysis on a graph based dataset
- **Task 2:** Create an efficient Spark program to calculate user influence
- **Bonus:** Use Azure Databricks to run Task 2

Twitter Social Graph Dataset

- tsv format
- Appx. 10GB of data (**do not download**)
- Edge list of (follower, followee) pairs
    - Directed
- # of followers distribution → power tail



Task 1 Exploratory Data Analysis

Two parts to Task 1

- Counting using Zeppelin notebook
  - Find the number of edges
  - Find the number of vertices

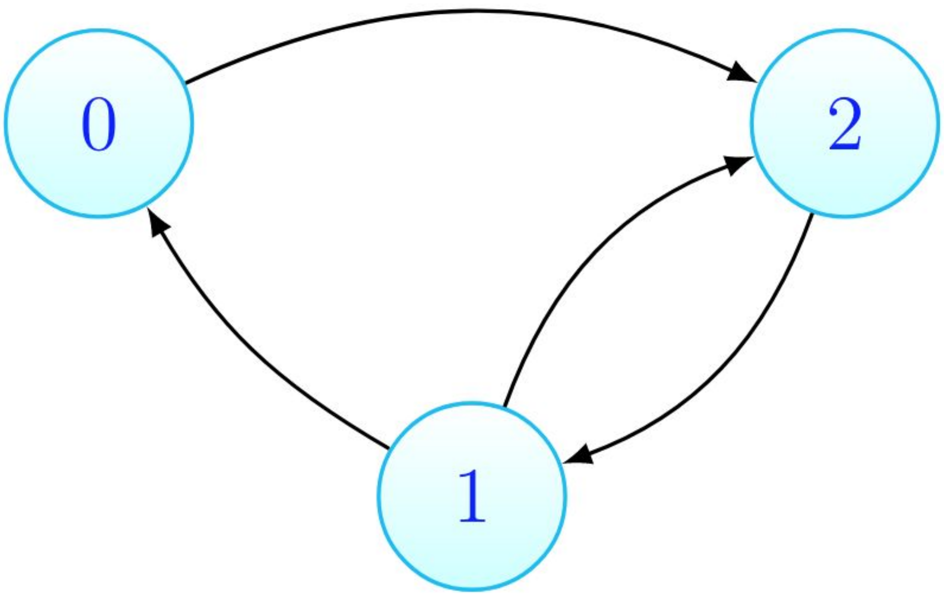
- Find top 100 most-popular users
  - RDD API
  - Spark DataFrame API

Task 2: PageRank

- Started as an algorithm to rank websites in search engine results
- Assign ranks based on the number of links pointing to them
- A page that has links from
  - Many nodes ⇒ high rank
  - A high-ranking node ⇒ (slightly less) high rank
- Implement Pagerank to find the rank of each user

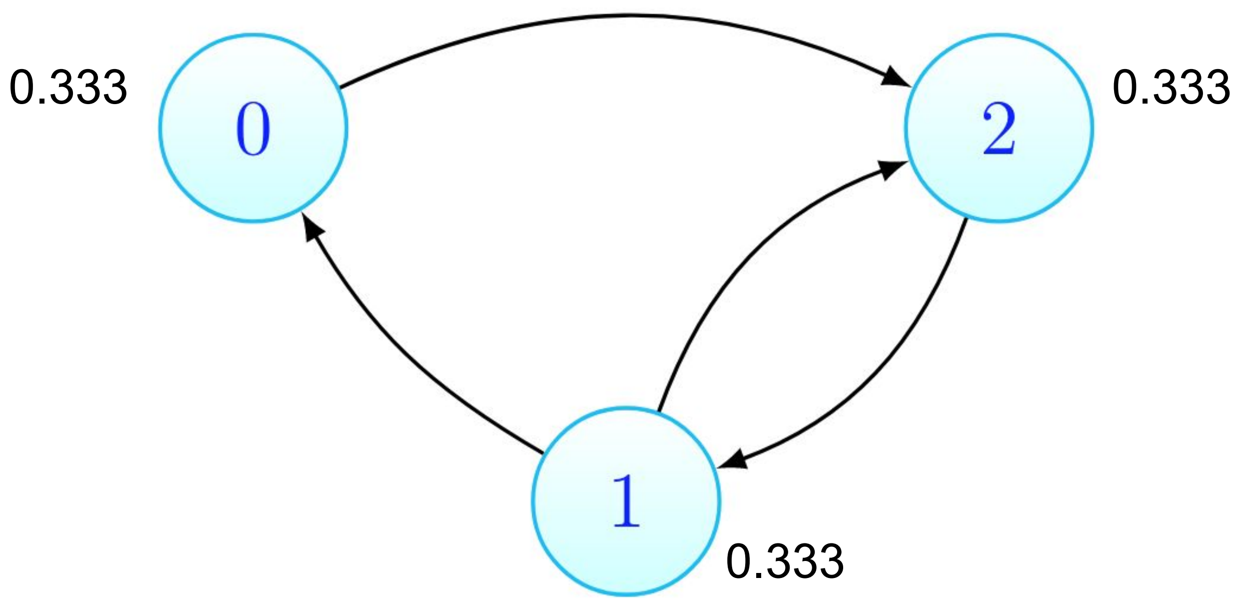
Basic PageRank

- How do we measure influence?
  - Intuitively, it should be the node with the most followers



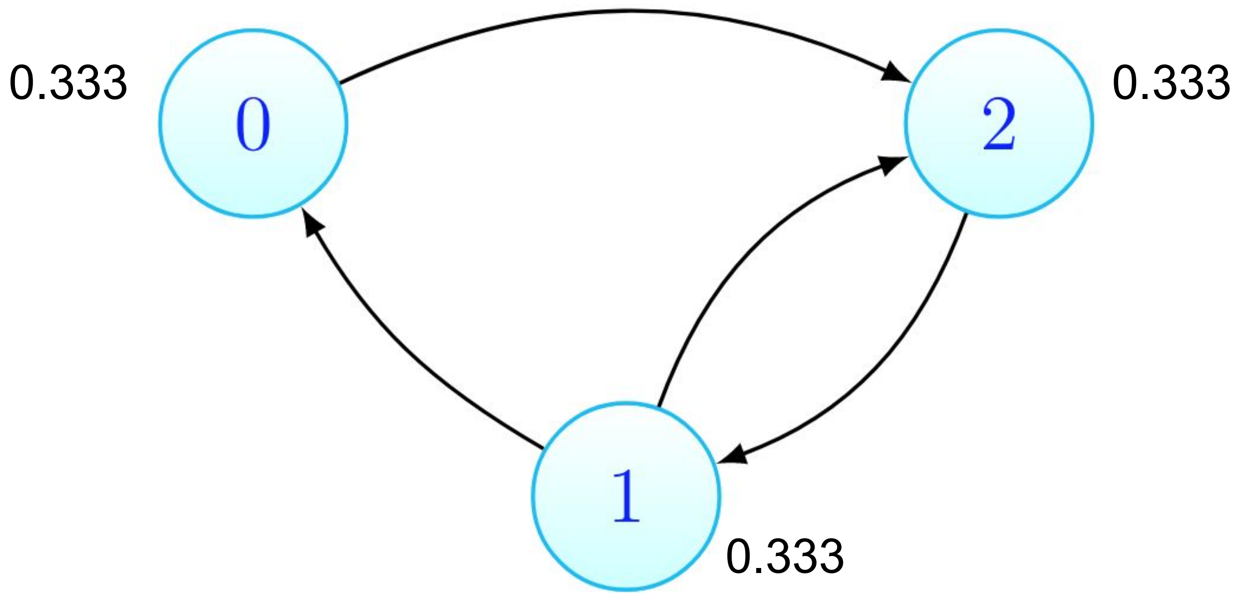
Basic PageRank

- Influence scores are initialized to -
  `1.0 / # of vertices`



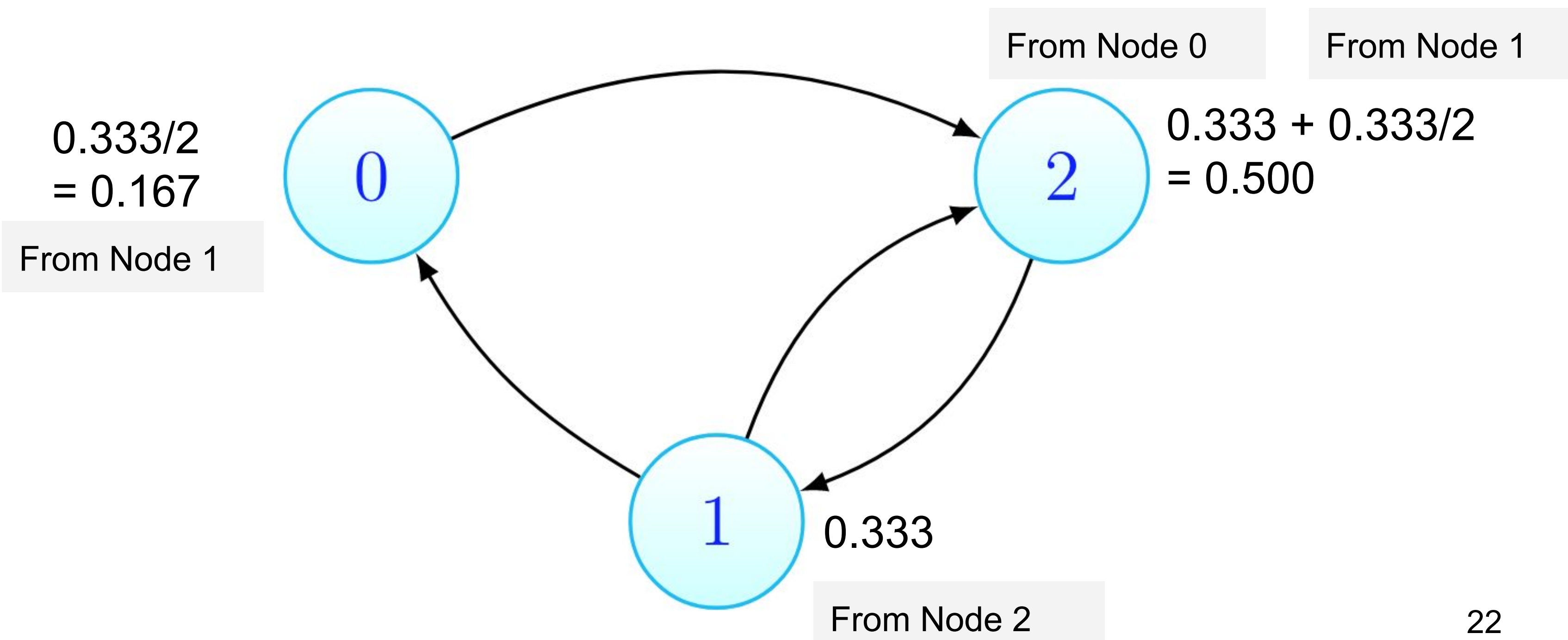
Basic PageRank

- In each iteration of the algorithm, scores of each user are redistributed between the users they are following



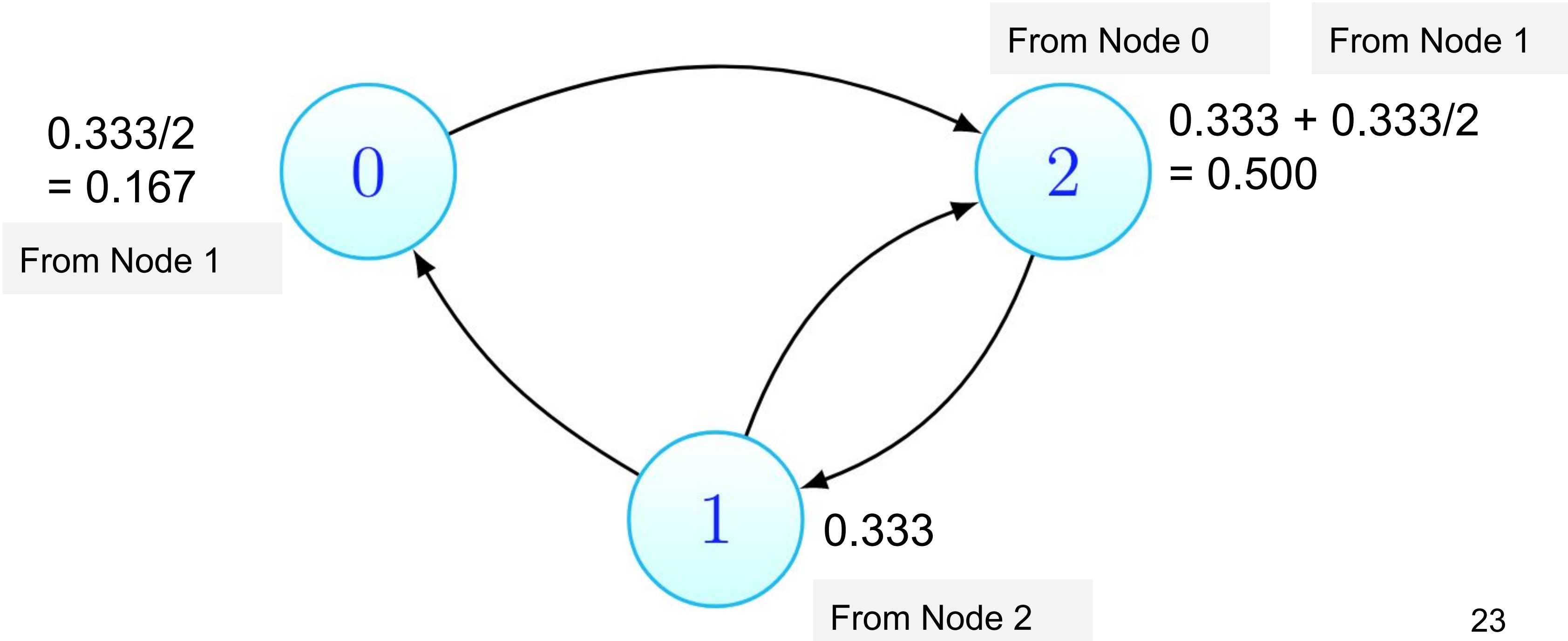
Basic PageRank

- In each iteration of the algorithm, scores of each user are redistributed between the users they are following



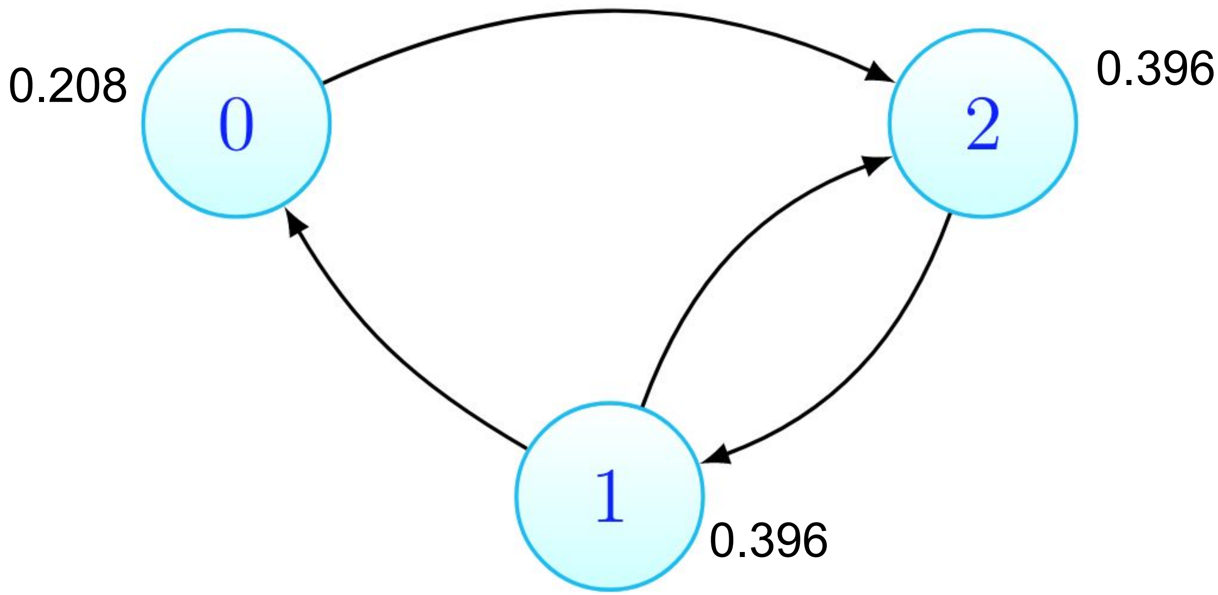
Basic PageRank

- Convergence is achieved when the scores of nodes do not change between iterations
- PageRank is guaranteed to converge



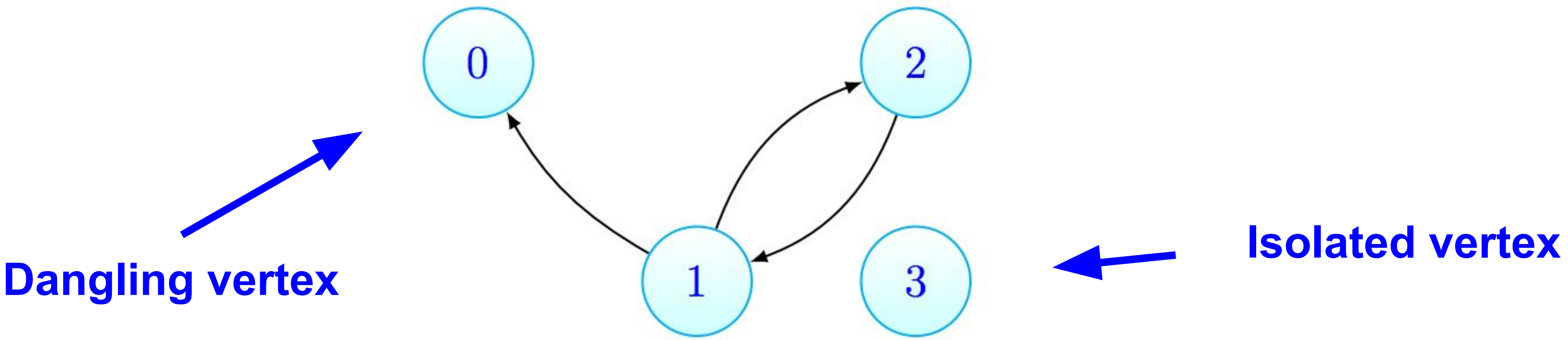
Basic PageRank

- Convergence is achieved when the scores of nodes do not change between iterations
- PageRank is guaranteed to converge



PageRank Terminology

- Dangling or sink vertex
  - No outgoing edges
  - Redistribute contribution equally among all vertices
- Isolated vertex
  - No incoming and outgoing edges
  - No isolated nodes in Project 4 dataset



PageRank Terminology

- Damping factor *d*
  - Represents the probability that a user clicking on links will continue clicking on them, traveling down an edge
  - Use $d = 0.85$

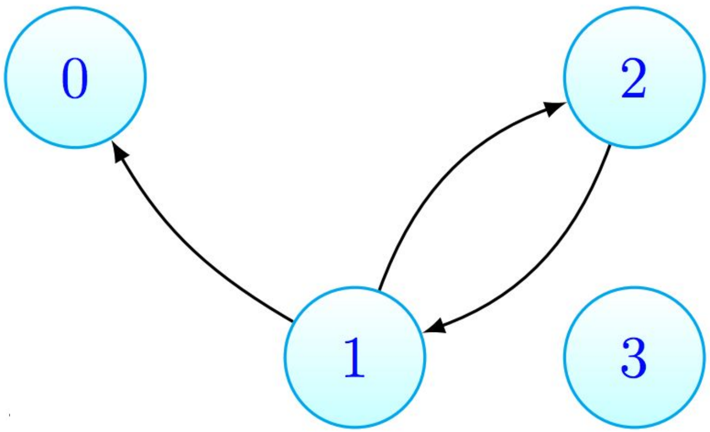
Visualizing Transitions

- Adjacency matrix:

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

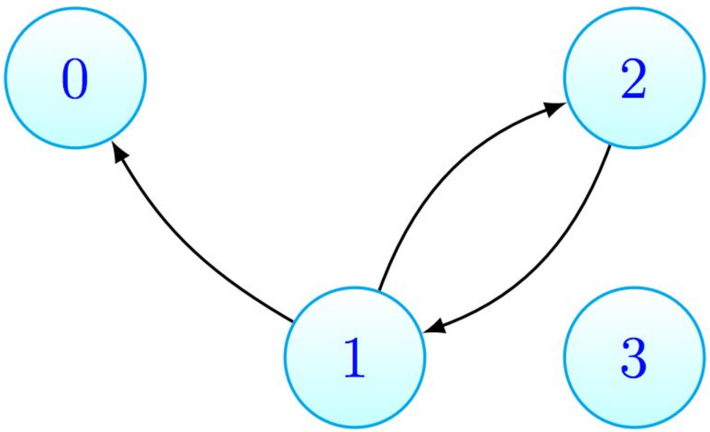
- Transition matrix: (rows sum to 1)

$$\mathbf{M} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$



$$M_{ij} = \frac{G_{ij}}{\sum_{k=1}^{n} G_{ik}} \left(\text{ when } \sum_{k=1}^{n} G_{ik} \neq 0 \right)$$

Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)}$$

d = 0.85

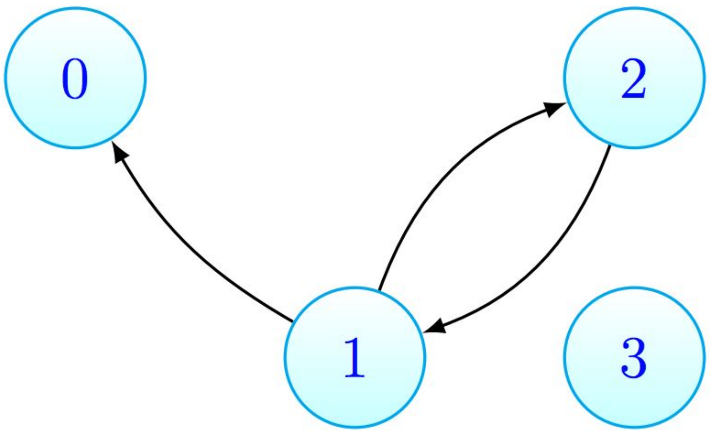
$$r_0^{(1)} = d(\frac{r_1^{(0)}}{2} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

$$r_1^{(1)} = d(\frac{r_2^{(0)}}{1} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

$$r_2^{(1)} = d(\frac{r_1^{(0)}}{2} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

$$r_3^{(1)} = d(\frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)}$$

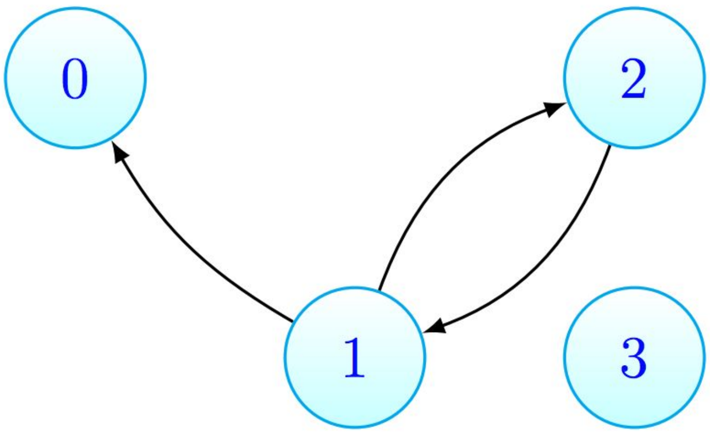
d = 0.85

Note: contributions from isolated and dangling vertices are constant in an iteration

Let

$$\epsilon = d\left(\frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}\right)$$

Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)}$$

d = 0.85

Note: contributions from isolated and dangling vertices are constant in an iteration

Let

$$\epsilon = d\left(\frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}\right)$$

This simplifies the formula

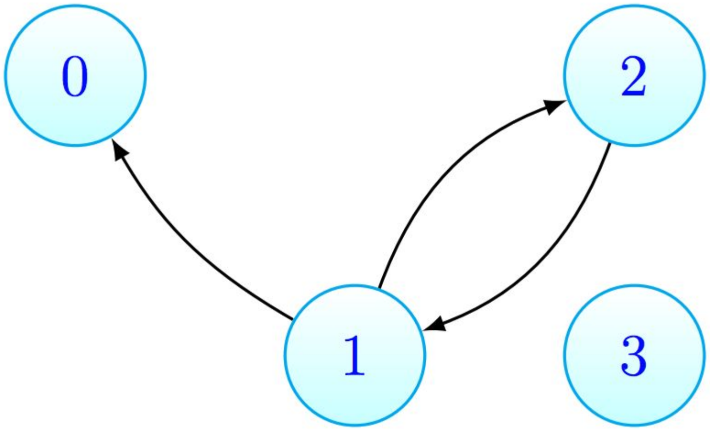
$$r_0^{(1)} = d\frac{r_1^{(0)}}{2} + \epsilon + (1-d)\frac{1}{n}$$

$$r_1^{(1)} = d\frac{r_2^{(0)}}{1} + \epsilon + (1-d)\frac{1}{n}$$

$$r_2^{(1)} = d\frac{r_1^{(0)}}{2} + \epsilon + (1-d)\frac{1}{n}$$

$$r_3^{(1)} = \epsilon + (1-d)\frac{1}{n}$$

Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)}$$

d = 0.85

$$\epsilon = 0.85 \times (0.25/4 + 0.25/4) = 0.106$$

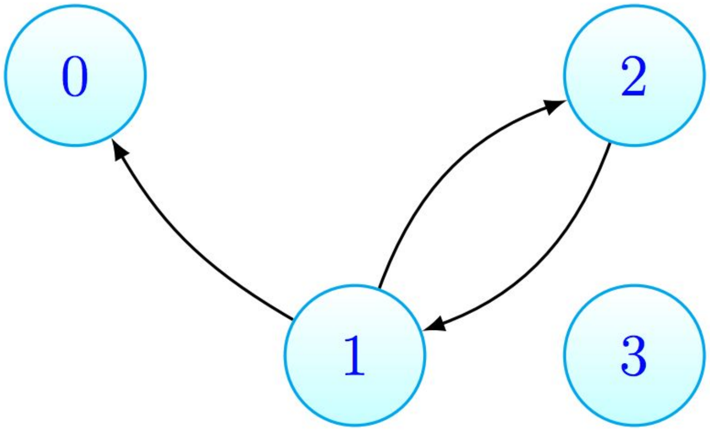
$$r_0^{(1)} = 0.85 \times 0.25/2 + 0.106 + 0.15 \times 0.25 = 0.25$$

$$r_1^{(1)} = 0.85 \times 0.25 + 0.106 + 0.15 \times 0.25 = 0.356$$

$$r_2^{(1)} = 0.85 \times 0.25/2 + 0.106 + 0.15 \times 0.25 = 0.25$$

$$r_3^{(1)} = 0.106 + 0.15 \times 0.25 = 0.144$$

Task 2: PageRank



Formula for calculating rank

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)}$$

d = 0.85

$$r_0^{(k)} = 0.2656$$

$$r_1^{(k)} = 0.3487$$

$$r_2^{(k)} = 0.2656$$

$$r_3^{(k)} = 0.1199$$

Basic PageRank Pseudocode

(Note: This does not meet the requirements of Task 2)

```
val links = spark.textFile(...).map(...).cache()
var ranks = // RDD of (URL, rank) pairs
for (i <- 1 to ITERATIONS)
{
    // Build an RDD of (targetURL, float) pairs
    // with the contributions sent by each page
    val contribs = links.join(ranks).flatMap
    {
        case (url, (links, rank)) =>
        links.map(dest => (dest, rank/links.size))
    }

    // Sum contributions by URL and get new ranks
    ranks = contribs.reduceByKey(_ + _)
                    .mapValues(sum => a/N + (1-a)*sum)
}
```

Reference: https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/SparkPageRank.scala

What you need to do for Task 2

- Run your page rank application on a 10GB graph data for *10 iterations*.
- Using HDInsight cluster on Azure:
  - Use the Terraform template provided for provisioning the cluster
  - **Very expensive - 2.6USD per hour**
- Scoring for Task 2 has 2 components:
  - 100% correctness for page rank - 30 points
  - Performance optimization (runtime within 30 minutes) - 30 points

Pagerank Hints

- Ensuring correctness
  - Make sure total scores sum to 1.0 in every iteration
  - Understand closures in Spark
    - Do not do something like this

```
val data = Array(1,2,3,4,5)
var counter = 0
var rdd = sc.parallelize(data)
rdd.foreach(x => counter += x)
println("Counter value: " + counter)
```

  - Graph representation
    - Adjacency lists use less memory than matrices
  - More detailed walkthroughs and sample calculations can be found [here]

Optimization Hints

- Understand RDD manipulations
  - Actions vs Transformations
  - Lazy transformations
- Use the Ambari UI
  - Are you utilizing your cluster completely? How can you change that? Refer optimization hints in the writeup.
- Use the Spark UI
  - Are your RDDs cached as expected?
  - Memory errors - check container logs
  - Parameter tuning applied successfully?
  - Exponential increase in partitions?
- How do you represent the node IDs? Int/String/Long?
- **Many more optimization hints in the writeup!**

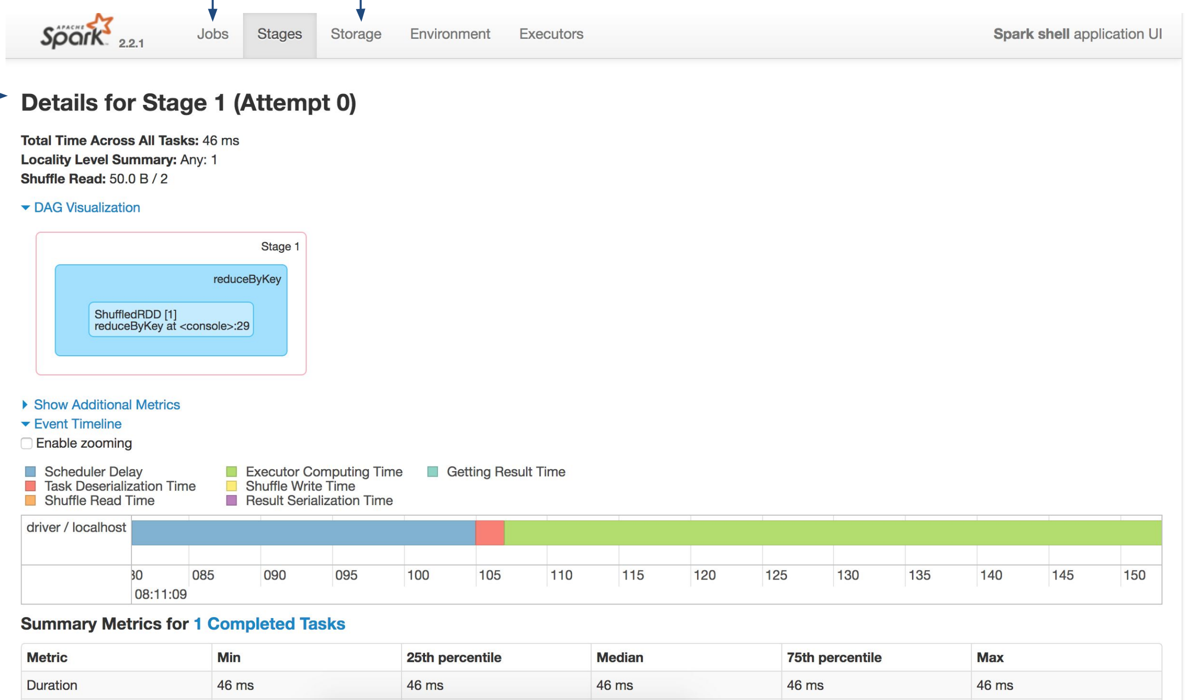
Spark UI

- Provides useful information on your Spark programs
- You can learn about resource utilization of your cluster
- Is a stepping stone to optimize your jobs

Status of RDD actions being computed

Info about cached RDDs and memory usage

In-depth job info



General Hints

- Starter code:
  - SparkUtils.scala - Use this for creating SparkSession objects.
- Test out commands on a Zeppelin notebook (refer to the Zeppelin primer)
- Test Driven Development (TDD):
  - Starter code contains a small graph test.
  - **Develop and test locally first!** HDInsight clusters are expensive
  - Add more test cases to check robustness.
  - Each submission can take anywhere from 6 min to an hour to run on the cluster.
- When in doubt, read the docs!
  - SparkSQL
  - RDD

Bonus Task - Databricks

- Databricks is an Apache Spark-based unified analytics platform.
- Azure Databricks is optimized for Azure
  - Software-as-a-Service
- One-click setup, an interactive workspace, and an optimized Databricks runtime
- Optimized connectors to Azure storage platforms for fast data access
- Run the same PageRank application (in Task 2) on Azure Databricks to compare the differences with Azure HDInsight

How to change your code?

```
object PageRank {
  def calculatePageRank(inputGraphPath: String, outputPath: String, iterations: Int, isLocal: Boolean): Unit = {
    val spark = SparkUtils.getSparkSession(isLocal, appName = "PageRank")
    val sc = spark.sparkContext

    … Your implementation goes here …
    graphRDD = sc.textFile(inputGraphPath)
    graphRDD.map(...)

    spark.close()
  }

  def main(args: Array[String]): Unit = {
    val inputGraph = "wasb://spark@cmuccpublicdatasets.blob.core.windows.net/Graph"
    val outputPath = "wasb:///pagerank-output"
    val iterations = 10

    calculatePageRank(inputGraph, outputPath, iterations, isLocal=false)
  }
}
```

How to change your code?

```
object PageRank {
 def calculatePageRank(inputGraphPath: String, outputPath: String, iterations: Int, isLocal: Boolean): Unit =
{
   val spark = SparkUtils.getSparkSession(isLocal, appName = "PageRank")
   val sc = spark.sparkContext

   val inputGraph = "wasb://spark@cmuccpublicdatasets.blob.core.windows.net/Graph"
   val outputPath = "dbfs:/pagerank-output"
   val iterations = 10
   … Your implementation goes here …
   graphRDD = sc.textFile(inputGraphPath)
   graphRDD.map(...)

   spark.close()
 }

 def main(args: Array[String]): Unit = {
   calculatePageRank(inputGraph, outputPath, iterations, isLocal=false)
 }
}
```

What you need to do for bonus?

- You can only get bonus (10 points) when:
  - 100% correctness
  - Runtime under 30 minutes on Databricks
- Copy your code to a Databricks notebook:
  - **Do not** create or destroy SparkSession objects
  - Change the output to DBFS instead of WASB
- Create a cluster and job using databricks-setup.sh
- Submitter takes in a job ID
- Don’t forget to destroy resources after you are done!

This Week

- **OLI, Unit 4: Cloud Storage**
  - Module 14: Cloud Storage
  - Module 15: Case Studies: Distributed File System
  - Module 16: Case Studies: NoSQL Databases
  - Module 17: Case Studies: Cloud Object Storage
- **Quiz 7 (OLI Module 14)**
  - Due **Friday**, March 18th, 2022, 11:59PM ET
- **Project 4 - Iterative processing with spark**
  - Due **Sunday**, March 27th, 2022, 11:59PM ET

Best Wishes on P4!!!

TEAM PROJECT
Twitter Data Analytics

Team Project Time Table

Phase	Deadline (<u>11:59PM ET</u>)
Phase 1 (20%) - **M1** - **M2** - **M3 (ckpt)**	• M1 CKPT (5%): Sun, 2/27 • M1 CKPT Report (5%) + Team Intro Form: Sun, 2/27 • M1 FINAL (10%): Sun, 3/6 • M2 CKPT (5%): Sun, 3/6 • **M2 FINAL (50%): Sun, 3/20** • **M3 CKPT (5%): Sun, 3/20** • Final Report + Code (20%): Tue, 3/22 **BONUSES:** • M1 Early Bird Bonus (5%): Sun, 2/27 • M2 Early Bird Bonus (5%): Sun, 3/6 • M2 Early Bird Bonus (5%): Sun, 3/6 • **M3 Early Bird Bonus (5%): Sun, 3/20** • **M3 Correctness Penalty Waiver: Sun, 3/20**

Suggested Tasks for Phase 1

Phase 1 weeks	**Tasks**	**Deadline**
Week 1-2 ● 02/14 - 02/27	● Team meeting ● Read Write Up & Report ● Complete M1 code & achieve correctness ● Start writing M2 solution ● Think about M3 database schema	● M1 Checkpoint due on 02/27 ● Checkpoint Report due on 02/27
Week 3 ● **02/28 - 03/06**	● Optimize for M1 performance ● Complete correct M2 code ● Start ETL process for M3	● M1 final target due on 03/06 ● M2 Checkpoint due on 03/06
Week 4-5 ● 03/07 - 03/20	● Optimize for M2 performance ● Finish M3 ETL process ● Complete M3 code & achieve correctness	● M2 final target due on 03/20 ● M3 Checkpoint due on 03/20 ● Final Report due on 03/22

Recap of M1 Performance

- Microservice 1
  - 55/69 teams achieved full score for M1

M1 Best Teams

Team	QRCode Throughput
CloudWatchers	155761.73
Random	125090.89
CaveMen	121991.08

Recap of M2 Performance

- Microservice 2
  - 49/69 teams had M2 checkpoint bonus
  - 53/69 teams made a non-zero score 600s submission
  - 37/69 teams achieved full scores for M2

M2 Best Teams

Team	Blockchain Throughput
ThreeCobblers	63287.62
ElasticPyjama	53172.53
MainframeComputing	51564.04

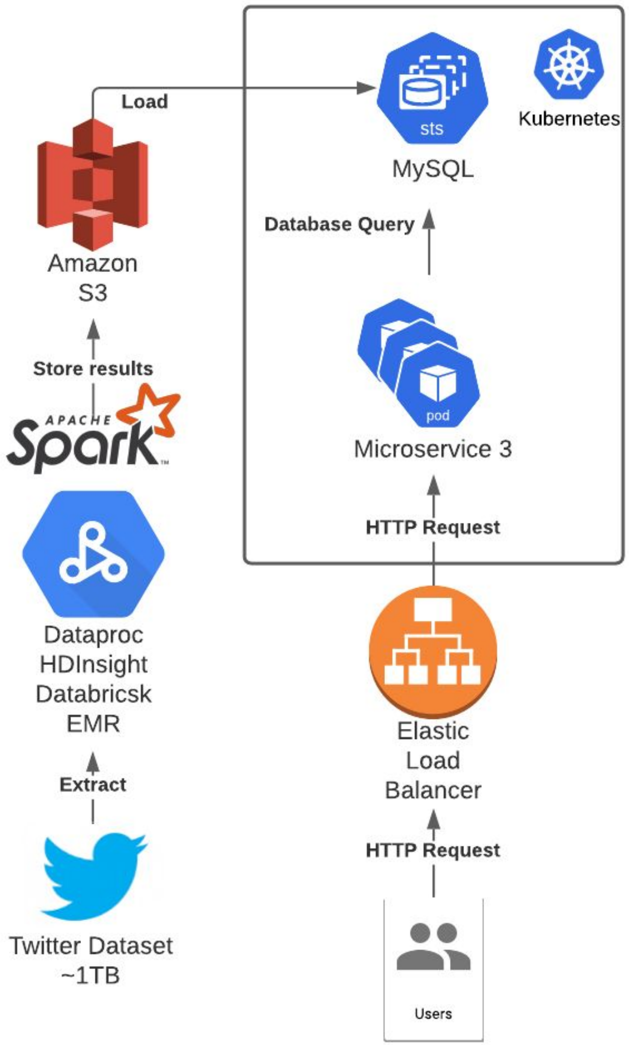
Recap of M3 Performance

- Microservice 3

Please start early!

Twitter Analytics System Architecture

- Building a performant web service
- Dealing with large scale real world tweet data
- HBase and MySQL optimization



Hourly Budget Reminder

- Your web service should not cost more than **$0.70/hour (if using MySQL) and $1.10/hour (if using HBase)**
- This includes:
    - EC2 cost (Even if you use spot instances, we will calculate your cost using the **on-demand** instance price)
    - **EBS cost**
    - **ELB cost - excluding LCU-hour cost**
    - We will not consider the cost of data transfer and EMR software
    - See writeup for details

Resource Constraint Reminder

- Self-managed Kubernetes cluster + optional EMR, consisting of M family instances **only**, smaller than or equal to **large** type
- MySQL must be installed on Kubernetes cluster
    - No standalone EC2 instance, no RDS
- Other types are allowed (e.g., t2.micro) **but only for testing**
    - Using these for live test submission = 100% penalty
- Only General Purpose (gp2) SSDs are allowed for storage
    - e.g **m5d is not allowed** since it uses NVMe storage
- AWS endpoints only (EC2/ELB).

Loading data & Backup

- Refer to MySQL Primer and Project 3 for data loading
  - P3 YetAnotherImportTsv can be helpful
  - Be very careful about escape characters
  - Be very careful about encodings
  - You can use temporary EC2 instance or EMR clusters to load your data
- Backup
  - For MySQL, make EBS snapshots of your data directory and attach it to your Pod
  - For HBase, you can backup and restore HBase database on S3 using the HBase snapshot

Hints

- Iterations rank higher than parameter tuning
  - Do not waste time tuning parameters when you have only one tenth of the target RPS!
  - Are all database queries necessary? Can they be done in your ETL pipelines instead?
  - **A good schema can easily double or even triple the throughput with no parameter tuning!**

- To do performance tuning, you first need to identify which part of your system is the bottleneck
  - Profile and monitor your system
  - Read the Profile Primer for profiling tools

Hints

- Web Tier
    - Concurrency model?
    - Connection pooling?
    - Caching result? (no third-party cache library!)
    - Is every computation in the web tier necessary?
        - Can they be done in ETL instead?
    - Have you optimized your code?
        - StringBuilder vs ‘+’
        - Try different library (gson vs Jackson vs jsoniter)

Hints

- Storage Tier - MySQL
  - Different MySQL engines
  - EBS I/O Credits and Burst Performance
- Storage Tier - HBase
  - Locality and compaction, region server split, etc
  - Scan can be really slow, try to avoid it if possible
    If you can’t, try to scan as few rows as possible
- Tune parameters ← Should be last thing to do!!
  - Check the official documentation
  - Search for performance tuning best practices

Best Wishes!!!